Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №4

з дисципліни

«Алгоритми та структури даних»

Виконав: студент групи ІМ-42

Максим Крамаренко Юрійович

номер варіанту: 17

Перевірив:

Сергієнко А. М.

Київ 2025

Постановка задачі

1. Представити напрямлений та ненапрямлений графи із заданими параметрами так само, як у лабораторній роботі №3.

Відмінність: коефіцієнт k = 1.0 - n3*0.01 - n4*0.01 - 0.3.

Отже, матриця суміжності Adir напрямленого графа за варіантом формується таким чином:

1) встановлюється параметр (seed) генератора випадкових чисел, рівне номеру варіанту n1n2n3n4;

2) матриця розміром n nзаповнюється згенерованими випадковими числами в діапазоні [0, 2.0);

3) обчислюється коефіцієнт k = 1.0 - n3*0.01 - n4*0.01 - 0.3, кожен елемент матриці множиться на коефіцієнт k;

4) елементи матриці округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.

2. Обчислити:

1) степені вершин напрямленого і ненапрямленого графів;

2) напівстепені виходу та заходу напрямленого графа;

3) чи є граф однорідним (регулярним), і якщо так, вказати степінь однорідності графа;

4) перелік висячих та ізольованих вершин.

3. Змінити матрицю Adir, коефіцієнт k = 1.0 - n3*0.005 - n4*0.005 - 0.27.

4. Для нового орграфа обчислити:

1) півстепені вершин;

2) всі шляхи довжини 2 і 3;

3) матрицю досяжності;

4) матрицю сильної зв’язності;

5) перелік компонент сильної зв’язності;

6) граф конденсації.

Результати вивести у графічне вікно, в консоль або файл.

Шляхи довжиною 2 і 3 слід шукати за матрицями A^2 і A^3, відповідно. Як результат вивести перелік шляхів, включно з усіма проміжними вершинами, через які проходить шлях.

Матрицю досяжності та компоненти сильної зв’язності слід шукати за допомогою операції транзитивного замикання. У переліку компонент слід вказати, які вершини належать до кожної компоненти.

Граф конденсації вивести у графічне вікно.

При проєктуванні програми слід врахувати наступне:

1) мова програмування обирається студентом самостійно;

2) графічне зображення усіх графів має формуватися програмою з тими ж вимогами, як у ЛР №3;

3) всі графи, включно із графом конденсації, обов’язково зображувати у графічному вікні;

4) типи та структури даних для внутрішнього представлення всіх даних у програмі слід вибрати самостійно;

5) обчислення перелічених у завданні результатів має виконуватися розробленою програмою (не вручну і не сторонніми засобами);

6) матриці, переліки степенів та маршрутів тощо можна виводити в графічне вікно або консоль — на розсуд студента;

7) у переліку знайдених шляхів треба вказувати не лише початок та кінець шляху, але й усі проміжні вершини, через які він проходить (наприклад, 1 – 5 – 3 – 2).

Варіант 17:

n1 = 4
n2 = 2
n3 = 1
n4 = 7

SEED = 4217

n = n3 + 10

k1 = 1.0 - n3*0.01 - n4*0.01 - 0.3
k2 = 1.0 - n3*0.005 - n4*0.005 - 0.27

Текст програми:

Файл №1 (main.py)

```
from matrix_utils import *
from graph_utils import *
from analyze_utils import *

# Define constants and parameters
n1 = 4
n2 = 2
n3 = 1
n4 = 7
# SEED = 4217

n = n3 + 10
```

```
# Coefficients for graph generation
k1 = 1.0 - n3*0.01 - n4*0.01 - 0.3
k2 = 1.0 - n3*0.005 - n4*0.005 - 0.27

# Function to analyze and print graph properties
def get_graph_info(dir, undir, modified=False):
    in_degrees = get_in_degrees(dir)
    out_degrees = get_out_degrees(dir)
    if not modified:
        # Print adjacency matrices
        print_matrix(dir, "Directed graph")
        print_matrix(undir, "Undirected graph")

        # Analyze degrees
        undir_degrees = get_undir_degrees(dir)
        print_undir_degrees(undir_degrees)

        print_dir_degrees(in_degrees, out_degrees)

        # Check if the graph is regular
        print(f"Graph is regular: {is_regular(undir_degrees)[0] if
is_regular(undir_degrees)[0] == False else is_regular(undir_degrees)}")

        # Identify isolated and leaf nodes
        isolated, leaf = get_isolated_and_leaf(undir_degrees)
        print(f"Isolated nodes: {isolated}")
        print(f"Leaf nodes: {leaf}")
        print("\n\n")
    else:
        # Print adjacency matrices
        print_matrix(dir, "New directed graph")
        print_matrix(undir, "New undirected graph")

        # Analyze degrees
        print_dir_degrees(in_degrees, out_degrees)



# === Original graph analysis ===
dir = get_dir(n, k1)  # Generate directed graph
undir = get_undir(dir)  # Convert to undirected graph
get_graph_info(dir, undir)
```

```python
# === Modified graph analysis ===
new_dir = get_dir(n, k2)  # Generate modified directed graph
new_undir = get_undir(new_dir)  # Convert to undirected graph
get_graph_info(new_dir, new_undir, modified=True)

# === Path analysis ===
# Analyze paths of specific lengths
paths_2 = paths_length_2(new_dir)
paths_3 = paths_length_3(new_dir)
print_paths(paths_2, 2)
print_paths(paths_3, 3)

# Compute reachability and strong connectivity
A = reachability_matrix(new_dir)
print_matrix(A, "Reachability matrix")

S = strong_connectivity_matrix(A)
print_matrix(S, "Strong connectivity")

# Identify strongly connected components
components = find_strong_components(S, n)
print("Strong connectivity components:")
for i, comp in enumerate(components):
    print(f"Component {i + 1}: nodes {comp}")

# Generate condensation graph
C = condensation_graph(components, new_dir)
print_matrix(C, "Condensation graph")

# === Draw graphs ===
draw_graph(dir, directed=True)  # Original directed graph
draw_graph(dir, directed=False)  # Original undirected graph
draw_graph(new_dir, directed=True)  # Modified directed graph
draw_graph(C, directed=True)  # Condensation graph
```

Файл №2 (matrix_utils.py**)**

```python
import math
import random

random.seed(4217)

def print_matrix(matrix, title="Matrix", line_length=1):
    print(f"\n=== {title} ===")
    for row in matrix:
```

```
        print(" ".join(f"{num:{line_length}}" for num in row))  # Each number is
line_length characters wide
    print()

def matrix_multiply(A, B):
    n = len(A)
    result = [[0] * n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            for k in range(n):
                result[i][j] += A[i][k] * B[k][j]
    return result

def matrix_add(A, B):
    n = len(A)
    result = [[0] * n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            result[i][j] = A[i][j] + B[i][j]
    return result

def get_dir(n, k):
    result = [[0] * n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            result[i][j] = math.floor(random.uniform(0, 2.0) * k)
    return result

def get_undir(dir):
    result = [[0] * len(dir) for _ in range(len(dir))]
    for i in range(len(dir)):
        for j in range(len(dir)):
            result[i][j] = result[j][i] = max(dir[i][j], dir[j][i])
    return result
```

Файл №3 (graph_utils.py)

```
import math
import random
import matplotlib.pyplot as plt

random.seed(4217)
```

```
def draw_graph(matrix, directed=False):
    R = 10  # Radius of the circular layout
    if (len(matrix) > 1):
        angle_step = 2 * math.pi / (len(matrix) - 1)  # Angle between nodes

    # Calculate node positions
    positions = []
    for i in range(len(matrix)):
        if i == 0:
            positions.append((0, 0))  # Center node
            continue
        angle = i * angle_step
        x = R * math.cos(angle)
        y = R * math.sin(angle)
        positions.append((x, y))

    node_R = 0.8  # Node radius

    # Plot the nodes
    plt.figure(figsize=(8, 8))
    for i, (x, y) in enumerate(positions):
        plt.scatter(x, y, s=500, color="gray", zorder=2)  # Draw node
        plt.text(x, y, str(i + 1), fontsize=12, ha="center", va="center", zorder=4)  #
Label node

    # Helper function to adjust for node boundary
    def adjust_for_R(x1, y1, x2, y2, offset):
        dx, dy = x2 - x1, y2 - y1
        length = math.sqrt(dx ** 2 + dy ** 2)
        if length == 0:
            return x1, y1, x2, y2
        scale = (length - offset) / length
        return x1 + dx * (1 - scale), y1 + dy * (1 - scale), x2 - dx * (1 - scale), y2 -
dy * (1 - scale)

    # Plot the edges
    m = 1
    def rand(a):
        return random.choice([random.uniform(-2*a,(node_R + a/8)),
random.uniform((node_R + a/8),2*a)])

    for i in range(len(matrix)):
        for j in range(len(matrix)):
```

```
            if matrix[i][j]:
                # Draw self-loop if a node connects to itself
                edge_color = (random.randint(0, 235) / 255, random.randint(0, 235) /
255, random.randint(0, 235) / 255)  # Generate a random RGB color
                if matrix[i][i]:
                    x, y = positions[i]
                    loop_radius = 1  # Adjust for better visibility
                    if(x != 0 and y != 0):
                        vector_length = math.sqrt(x ** 2 + y ** 2)
                        x += x * loop_radius / vector_length
                        y += y * loop_radius / vector_length
                    else:
                        y += loop_radius
                    loop = plt.Circle((x, y), loop_radius, color=edge_color, fill=False,
zorder=1)
                    plt.gca().add_patch(loop)
                x1, y1 = positions[i]
                x2, y2 = positions[j]

                dx, dy = x2 - x1, y2 - y1
                length = math.sqrt(dx ** 2 + dy ** 2)
                x1, y1, x2, y2 = adjust_for_R(x1, y1, x2, y2, node_R)
                if (length >= 2*(R - node_R) and length <= 2*(R + node_R)):  # If the
line goes through the center
                    midx = (x1 + x2) / 2 + rand(m)
                    midy = (y1 + y2) / 2 + rand(m)
                    # Draw directed edge (arrow)
                    if directed:
                        plt.plot([x1, midx], [y1, midy], color=edge_color, zorder=3)
                        plt.arrow(midx, midy, x2 - midx, y2 - midy, head_width=0.30,
length_includes_head=True, color=edge_color, zorder=4)
                        continue
                    plt.plot([x1, midx, x2], [y1, midy, y2], color=edge_color, zorder=3)
                    continue
                if directed:
                    plt.arrow(x1, y1, x2 - x1, y2 - y1, head_width=0.30,
length_includes_head=True, color=edge_color, zorder=4)
                    continue
                plt.plot([x1, x2], [y1, y2], color=edge_color, zorder=3)

    # Set plot limits and hide axes
    plt.xlim(-15, 15)
    plt.ylim(-15, 15)
    plt.axis("off")
```

```
    plt.show()
```

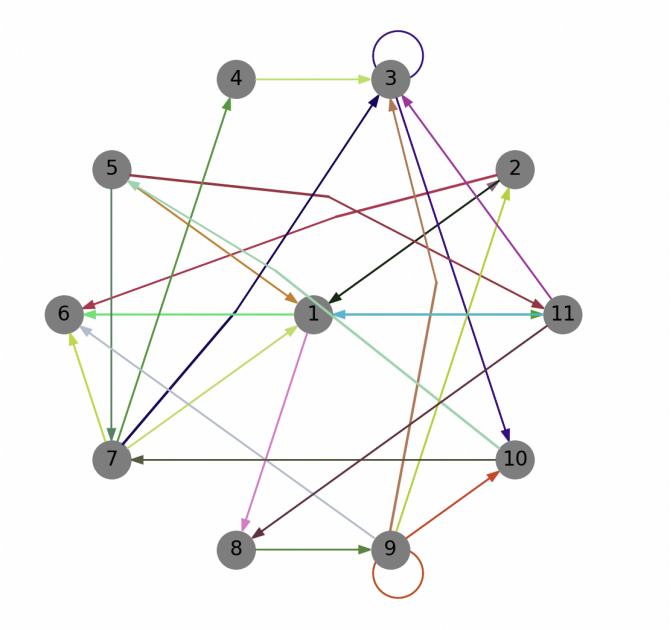
Файл №4 (analyze_utils.py)

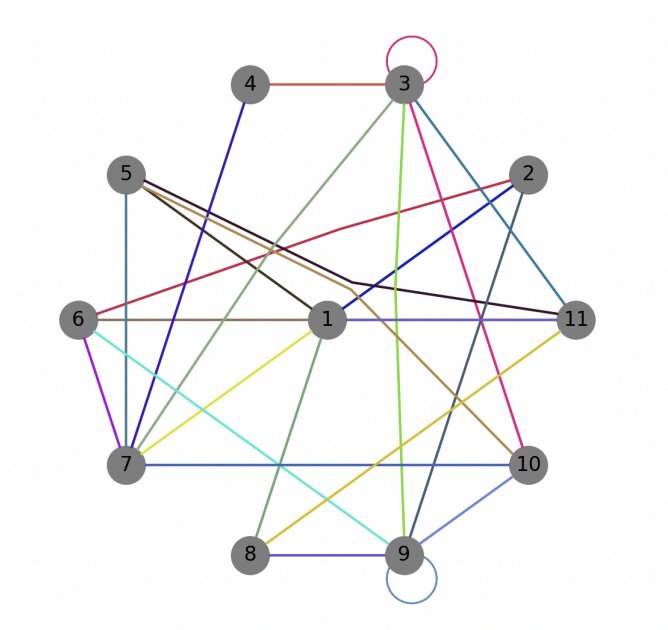
```
from matrix_utils import matrix_multiply, matrix_add

def get_undir_degrees(dir):
    return {i: (sum(dir[i]) + sum(dir[j][i] for j in range(len(dir)))) for i in
range(len(dir))}

def print_undir_degrees(undir_degree):
    print("\n\n=== Node degrees of undirected graph ===\n")
    print(f"{'== Node ==':^10} {'== Degree ==':^15}\n")
    for index, value in undir_degree.items():
        print(f"{(index + 1):^10}->{value:^15}")
    print()

def get_in_degrees(dir):
    return {i: sum(dir[j][i] for j in range(len(dir))) for i in range(len(dir))}

def get_out_degrees(dir):
    return {i: sum(dir[i]) for i in range(len(dir))}

def print_dir_degrees(in_degree, out_degree):
    print("\n\n=== Node degrees of directed graph ===\n")
    print(f"{'== Node ==':^10}  {'== In-Degree ==':^15}  {'== Out-Degree ==':^15}\n")
    for i in range(len(in_degree)):
        print(f"{(i + 1):^10}->{in_degree[i]:^15}  {out_degree[i]:^15}")
    print()

def is_regular(undir_degrees):
    for i in range(len(undir_degrees)):
        if (undir_degrees[0] != undir_degrees[i]): return (False, 0)
    return (True, undir_degrees[0])

def get_isolated_and_leaf(matrix):
    isolated = []
    leaf = []
    for i in range(len(matrix)):
        if matrix[i] == 0: isolated.append(i + 1)
        if matrix[i] == 1: leaf.append(i + 1)
    return isolated, leaf
```

```
def paths_length_2(matrix):
    A2 = matrix_multiply(matrix, matrix)
    paths = []

    for i in range(len(matrix)):
        for j in range(len(matrix)):
            if A2[i][j] > 0:
                for k in range(len(matrix)):
                    if matrix[i][k] and matrix[k][j]:
                        paths.append(f"{i+1} -> {k+1} -> {j+1}")

    return paths

def paths_length_3(matrix):
    A3 = matrix_multiply(matrix, matrix_multiply(matrix, matrix))
    paths = []

    for i in range(len(matrix)):
        for k in range(len(matrix)):
            if A3[i][k] > 0:
                for l in range(len(matrix)):
                    for j in range(len(matrix)):
                        if matrix[i][j] and matrix[j][l] and matrix[l][k]:
                            paths.append(f"{i+1} -> {j+1} -> {l+1} -> {k+1}")

    return paths

def print_paths(paths, length):
    if not paths:  # Check if paths list is empty
        print(f"\nNo paths of length {length} found.")
        return

    print(f"\nPaths of length {length}:")
    line_length = 6  # Number of paths per line
    max_width = max(len(f"({path})") for path in paths)  # Find the widest path

    for i in range(0, len(paths), line_length):
        formatted_paths = [f"({path})".ljust(max_width) for path in paths[i:i +
line_length]]
        print(", ".join(formatted_paths) + ",")
    print()

def reachability_matrix(matrix):
    temp = matrix
```

```
    reach = matrix

    for _ in range(len(matrix) - 1):
        temp = matrix_multiply(temp, matrix)
        reach = matrix_add(reach, temp)

    for i in range(len(matrix)):
        for j in range(len(matrix)):
            if reach[i][j] > 0:
                reach[i][j] = 1
    return reach

import numpy as np
def strong_connectivity_matrix(A):
    R = np.array(A)
    return np.logical_and(R, R.T).astype(int)

def find_strong_components(S, n):
    visited = [False] * n
    components = []

    for v in range(n):
        if not visited[v]:
            component = []
            for u in range(n):
                if S[v, u] == 1:
                    component.append(u + 1)
                    visited[u] = True
            if component:
                components.append(component)
    return components

def condensation_graph(components, dir_matrix):
    condensation = [[0 for _ in range(len(components))] for _ in range(len(components))]
    for i, comp_i in enumerate(components):
        for j, comp_j in enumerate(components):
            if i != j:
                for v in comp_i:
                    for u in comp_j:
                        if dir_matrix[v - 1][u - 1] == 1:
                            condensation[i][j] = 1
                            break
    return condensation
```

Тестування програми:






```
=== Directed graph ===
0 1 0 0 0 1 0 1 0 0 1
1 0 0 0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0
1 0 1 1 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0
0 1 1 0 0 1 0 0 1 1 0
0 0 0 0 1 0 1 0 0 0 0
1 0 1 0 0 0 0 1 0 0 0
```

```
=== Undirected graph ===
0 1 0 0 1 1 1 1 0 0 1
1 0 0 0 0 1 0 0 1 0 0
0 0 1 1 0 0 1 0 1 1 1
0 0 1 0 0 0 1 0 0 0 0
1 0 0 0 0 0 1 0 0 1 1
1 1 0 0 0 0 1 0 1 0 0
1 0 1 1 1 1 0 0 0 1 0
1 0 0 0 0 0 0 0 1 0 1
0 1 1 0 0 1 0 1 1 1 0
0 0 1 0 1 0 1 0 1 0 0
1 0 1 0 1 0 0 1 0 0 0
```

Степені матриці

```
=== Node degrees of undirected graph ===

== Node ==  == Degree ==

    1     ->      8
    2     ->      4
    3     ->      7
    4     ->      2
    5     ->      4
    6     ->      4
    7     ->      6
    8     ->      3
    9     ->      7
    10    ->      4
    11    ->      5
```

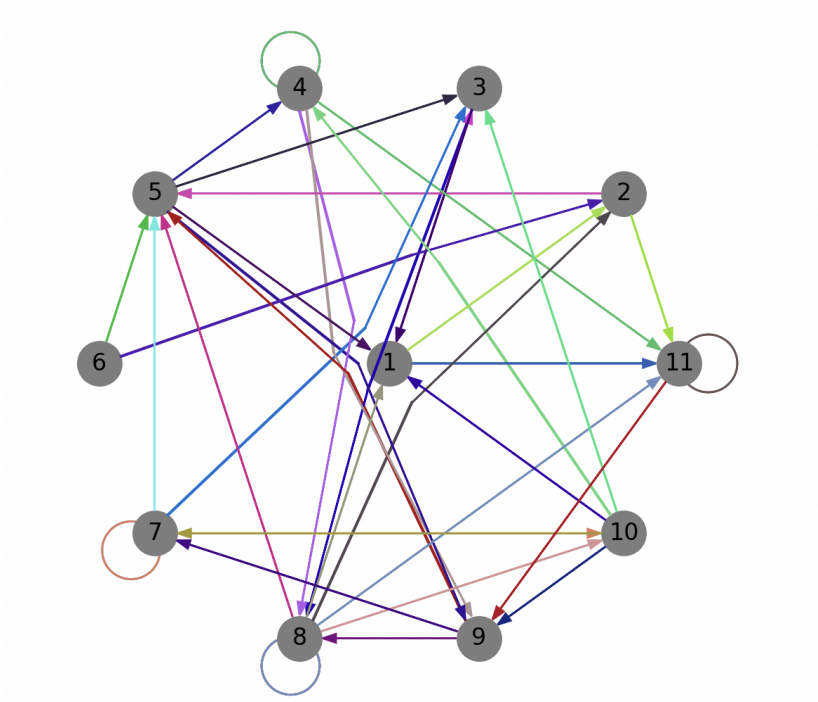
Напівстепені матриці

```
=== Node degrees of directed graph ===

== Node ==  == In-Degree ==  == Out-Degree ==

    1     ->      4               4
    2     ->      2               2
    3     ->      5               2
    4     ->      1               1
    5     ->      1               3
    6     ->      4               0
    7     ->      2               4
    8     ->      2               1
    9     ->      2               5
    10    ->      2               2
    11    ->      2               3
```

Однорідність графа, ізольовані та висячі вершини:

```
Graph is regular: False
Isolated nodes: []
Leaf nodes: []
```

Матриця зміненого коефіцієнта (k):




```
=== New directed graph ===
0 1 1 0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0 0 0 1
1 0 0 0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 1 1 0 1
1 0 1 1 0 0 0 0 1 0 0
0 1 0 0 1 0 0 0 0 0 0
0 0 1 0 1 0 1 0 0 1 0
1 1 0 0 1 0 0 1 0 1 1
0 0 0 0 1 0 1 1 0 0 0
1 0 1 1 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0 1 0 1
```

```
=== New undirected graph ===
0 1 1 0 1 0 0 1 0 1 1
1 0 0 0 1 1 0 1 0 0 1
1 0 0 0 1 0 1 1 0 1 0
0 0 0 1 1 0 0 1 1 1 1
1 1 1 1 0 1 1 1 1 0 0
0 1 0 0 1 0 0 0 0 0 0
0 0 1 0 1 0 1 0 1 1 0
1 1 1 1 1 0 0 1 1 1 1
0 0 0 1 1 0 1 1 0 1 1
1 0 1 1 0 0 1 1 1 0 0
1 1 0 1 0 0 0 1 1 0 1
```

Півстепені вершин:

```
=== Node degrees of directed graph ===

== Node ==   == In-Degree ==   == Out-Degree ==

    1     ->       4                3
    2     ->       3                2
    3     ->       4                2
    4     ->       3                4
    5     ->       5                4
    6     ->       0                2
    7     ->       3                4
    8     ->       4                6
    9     ->       4                3
    10    ->       2                5
    11    ->       5                2
```

Всі шляхи довжиною 2:

```
Paths of length 2:
(1 -> 3 -> 1)   , (1 -> 2 -> 5)   , (1 -> 3 -> 8)   , (1 -> 11 -> 9)  , (1 -> 2 -> 11)  , (1 -> 11 -> 11) ,
(2 -> 5 -> 1)   , (2 -> 5 -> 3)   , (2 -> 5 -> 4)   , (2 -> 5 -> 9)   , (2 -> 11 -> 9)  , (2 -> 11 -> 11) ,
(3 -> 8 -> 1)   , (3 -> 1 -> 2)   , (3 -> 8 -> 2)   , (3 -> 1 -> 3)   , (3 -> 8 -> 5)   , (3 -> 8 -> 8)   ,
(3 -> 8 -> 10)  , (3 -> 1 -> 11)  , (3 -> 8 -> 11)  , (4 -> 8 -> 1)   , (4 -> 8 -> 2)   , (4 -> 4 -> 4)   ,
(4 -> 8 -> 5)   , (4 -> 9 -> 5)   , (4 -> 9 -> 7)   , (4 -> 4 -> 8)   , (4 -> 8 -> 8)   , (4 -> 9 -> 8)   ,
(4 -> 4 -> 9)   , (4 -> 11 -> 9)  , (4 -> 8 -> 10)  , (4 -> 4 -> 11)  , (4 -> 8 -> 11)  , (4 -> 11 -> 11) ,
(5 -> 3 -> 1)   , (5 -> 1 -> 2)   , (5 -> 1 -> 3)   , (5 -> 4 -> 4)   , (5 -> 9 -> 5)   , (5 -> 9 -> 7)   ,
(5 -> 3 -> 8)   , (5 -> 4 -> 8)   , (5 -> 9 -> 8)   , (5 -> 4 -> 9)   , (5 -> 1 -> 11)  , (5 -> 4 -> 11)  ,
(6 -> 5 -> 1)   , (6 -> 5 -> 3)   , (6 -> 5 -> 4)   , (6 -> 2 -> 5)   , (6 -> 5 -> 9)   , (6 -> 2 -> 11)  ,
(7 -> 3 -> 1)   , (7 -> 5 -> 1)   , (7 -> 10 -> 1)  , (7 -> 5 -> 3)   , (7 -> 7 -> 3)   , (7 -> 10 -> 3)  ,
(7 -> 5 -> 4)   , (7 -> 10 -> 4)  , (7 -> 7 -> 5)   , (7 -> 7 -> 7)   , (7 -> 10 -> 7)  , (7 -> 3 -> 8)   ,
(7 -> 5 -> 9)   , (7 -> 10 -> 9)  , (7 -> 7 -> 10)  , (8 -> 5 -> 1)   , (8 -> 8 -> 1)   , (8 -> 10 -> 1)  ,
(8 -> 1 -> 2)   , (8 -> 8 -> 2)   , (8 -> 1 -> 3)   , (8 -> 5 -> 3)   , (8 -> 10 -> 3)  , (8 -> 5 -> 4)   ,
(8 -> 10 -> 4)  , (8 -> 2 -> 5)   , (8 -> 8 -> 5)   , (8 -> 10 -> 7)  , (8 -> 8 -> 8)   , (8 -> 5 -> 9)   ,
(8 -> 10 -> 9)  , (8 -> 11 -> 9)  , (8 -> 8 -> 10)  , (8 -> 1 -> 11)  , (8 -> 2 -> 11)  , (8 -> 8 -> 11)  ,
(8 -> 11 -> 11) , (9 -> 5 -> 1)   , (9 -> 8 -> 1)   , (9 -> 8 -> 2)   , (9 -> 5 -> 3)   , (9 -> 7 -> 3)   ,
(9 -> 5 -> 4)   , (9 -> 7 -> 5)   , (9 -> 8 -> 5)   , (9 -> 7 -> 7)   , (9 -> 8 -> 8)   , (9 -> 5 -> 9)   ,
(9 -> 7 -> 10)  , (9 -> 8 -> 10)  , (9 -> 8 -> 11)  , (10 -> 3 -> 1)  , (10 -> 1 -> 2)  , (10 -> 1 -> 3)  ,
(10 -> 7 -> 3)  , (10 -> 4 -> 4)  , (10 -> 7 -> 5)  , (10 -> 9 -> 5)  , (10 -> 7 -> 7)  , (10 -> 9 -> 7)  ,
(10 -> 3 -> 8)  , (10 -> 4 -> 8)  , (10 -> 9 -> 8)  , (10 -> 4 -> 9)  , (10 -> 7 -> 10) , (10 -> 1 -> 11) ,
(10 -> 4 -> 11) , (11 -> 9 -> 5)  , (11 -> 9 -> 7)  , (11 -> 9 -> 8)  , (11 -> 11 -> 9) , (11 -> 11 -> 11),
```

Всі шляхи довжиною 3:

```
Paths of length 3:
(1 -> 2 -> 5 -> 1)    , (1 -> 3 -> 8 -> 1)    , (1 -> 3 -> 1 -> 2)    , (1 -> 3 -> 8 -> 2)    , (1 -> 3 -> 1 -> 3)    , (1 -> 2 -> 5 -> 3)    ,
(1 -> 2 -> 5 -> 4)    , (1 -> 3 -> 8 -> 5)    , (1 -> 11 -> 9 -> 5)   , (1 -> 11 -> 9 -> 7)   , (1 -> 3 -> 8 -> 8)    , (1 -> 11 -> 9 -> 8)   ,
(1 -> 2 -> 5 -> 9)    , (1 -> 2 -> 11 -> 9)   , (1 -> 11 -> 11 -> 9)  , (1 -> 3 -> 8 -> 10)  , (1 -> 3 -> 1 -> 11)  , (1 -> 3 -> 8 -> 11)  ,
(1 -> 2 -> 11 -> 11)  , (1 -> 11 -> 11 -> 11) , (2 -> 5 -> 3 -> 1)    , (2 -> 5 -> 1 -> 2)    , (2 -> 5 -> 1 -> 3)    , (2 -> 5 -> 4 -> 4)    ,
(2 -> 5 -> 9 -> 5)    , (2 -> 11 -> 9 -> 5)   , (2 -> 5 -> 9 -> 7)    , (2 -> 11 -> 9 -> 7)   , (2 -> 5 -> 3 -> 8)    , (2 -> 5 -> 4 -> 8)    ,
(2 -> 5 -> 9 -> 8)    , (2 -> 11 -> 9 -> 8)   , (2 -> 5 -> 4 -> 9)    , (2 -> 11 -> 11 -> 9)  , (2 -> 5 -> 1 -> 11)  , (2 -> 5 -> 4 -> 11)  ,
(2 -> 11 -> 11 -> 11) , (3 -> 1 -> 3 -> 1)    , (3 -> 8 -> 5 -> 1)    , (3 -> 8 -> 8 -> 1)    , (3 -> 8 -> 10 -> 1)   , (3 -> 8 -> 1 -> 2)    ,
(3 -> 8 -> 8 -> 2)    , (3 -> 8 -> 1 -> 3)    , (3 -> 8 -> 5 -> 3)    , (3 -> 8 -> 10 -> 3)   , (3 -> 8 -> 5 -> 4)    , (3 -> 8 -> 10 -> 4)   ,
(3 -> 1 -> 2 -> 5)    , (3 -> 8 -> 2 -> 5)    , (3 -> 8 -> 8 -> 5)    , (3 -> 8 -> 10 -> 7)   , (3 -> 1 -> 3 -> 8)    , (3 -> 8 -> 8 -> 8)    ,
(3 -> 8 -> 5 -> 9)    , (3 -> 8 -> 10 -> 9)   , (3 -> 1 -> 11 -> 9)   , (3 -> 8 -> 11 -> 9)   , (3 -> 8 -> 8 -> 10)   , (3 -> 8 -> 1 -> 11)   ,
(3 -> 1 -> 2 -> 11)   , (3 -> 8 -> 2 -> 11)   , (3 -> 8 -> 8 -> 11)   , (3 -> 1 -> 11 -> 11)  , (3 -> 8 -> 11 -> 11)  , (4 -> 8 -> 5 -> 1)    ,
(4 -> 9 -> 5 -> 1)    , (4 -> 4 -> 8 -> 1)    , (4 -> 8 -> 8 -> 1)    , (4 -> 9 -> 8 -> 1)    , (4 -> 8 -> 10 -> 1)   , (4 -> 8 -> 1 -> 2)    ,
(4 -> 4 -> 8 -> 2)    , (4 -> 8 -> 8 -> 2)    , (4 -> 9 -> 8 -> 2)    , (4 -> 8 -> 1 -> 3)    , (4 -> 8 -> 5 -> 3)    , (4 -> 9 -> 5 -> 3)    ,
(4 -> 9 -> 7 -> 3)    , (4 -> 8 -> 10 -> 3)   , (4 -> 4 -> 4 -> 4)    , (4 -> 8 -> 5 -> 4)    , (4 -> 9 -> 5 -> 4)    , (4 -> 8 -> 10 -> 4)   ,
(4 -> 8 -> 2 -> 5)    , (4 -> 9 -> 7 -> 5)    , (4 -> 4 -> 8 -> 5)    , (4 -> 8 -> 8 -> 5)    , (4 -> 9 -> 8 -> 5)    , (4 -> 4 -> 9 -> 5)    ,
(4 -> 11 -> 9 -> 5)   , (4 -> 9 -> 7 -> 7)    , (4 -> 4 -> 9 -> 7)    , (4 -> 11 -> 9 -> 7)   , (4 -> 8 -> 10 -> 7)   , (4 -> 4 -> 4 -> 8)    ,
(4 -> 4 -> 8 -> 8)    , (4 -> 8 -> 8 -> 8)    , (4 -> 9 -> 8 -> 8)    , (4 -> 4 -> 9 -> 8)    , (4 -> 11 -> 9 -> 8)   , (4 -> 4 -> 4 -> 9)    ,
(4 -> 8 -> 5 -> 9)    , (4 -> 9 -> 5 -> 9)    , (4 -> 8 -> 10 -> 9)   , (4 -> 4 -> 11 -> 9)   , (4 -> 8 -> 11 -> 9)   , (4 -> 11 -> 11 -> 9)  ,
(4 -> 9 -> 7 -> 10)   , (4 -> 4 -> 8 -> 10)   , (4 -> 8 -> 8 -> 10)   , (4 -> 9 -> 8 -> 10)   , (4 -> 8 -> 1 -> 11)   , (4 -> 8 -> 2 -> 11)   ,
(4 -> 4 -> 4 -> 11)   , (4 -> 4 -> 8 -> 11)   , (4 -> 8 -> 8 -> 11)   , (4 -> 9 -> 8 -> 11)   , (4 -> 4 -> 11 -> 11)  , (4 -> 8 -> 11 -> 11)  ,
(4 -> 11 -> 11 -> 11) , (5 -> 1 -> 3 -> 1)    , (5 -> 9 -> 5 -> 1)    , (5 -> 3 -> 8 -> 1)    , (5 -> 4 -> 8 -> 1)    , (5 -> 9 -> 8 -> 1)    ,
(5 -> 3 -> 1 -> 2)    , (5 -> 3 -> 8 -> 2)    , (5 -> 4 -> 8 -> 2)    , (5 -> 9 -> 8 -> 2)    , (5 -> 3 -> 1 -> 3)    , (5 -> 9 -> 5 -> 3)    ,
(5 -> 9 -> 7 -> 3)    , (5 -> 4 -> 4 -> 4)    , (5 -> 9 -> 5 -> 4)    , (5 -> 1 -> 2 -> 5)    , (5 -> 9 -> 7 -> 5)    , (5 -> 3 -> 8 -> 5)    ,
(5 -> 4 -> 8 -> 5)    , (5 -> 9 -> 8 -> 5)    , (5 -> 4 -> 9 -> 5)    , (5 -> 9 -> 7 -> 7)    , (5 -> 4 -> 9 -> 7)    , (5 -> 1 -> 3 -> 8)    ,
(5 -> 4 -> 4 -> 8)    , (5 -> 3 -> 8 -> 8)    , (5 -> 4 -> 8 -> 8)    , (5 -> 9 -> 8 -> 8)    , (5 -> 4 -> 9 -> 8)    , (5 -> 4 -> 4 -> 9)    ,
(5 -> 9 -> 5 -> 9)    , (5 -> 1 -> 11 -> 9)   , (5 -> 4 -> 11 -> 9)   , (5 -> 9 -> 7 -> 10)   , (5 -> 3 -> 8 -> 10)   , (5 -> 4 -> 8 -> 10)   ,
(5 -> 9 -> 8 -> 10)   , (5 -> 3 -> 1 -> 11)   , (5 -> 1 -> 2 -> 11)   , (5 -> 4 -> 4 -> 11)   , (5 -> 3 -> 8 -> 11)   , (5 -> 4 -> 8 -> 11)   ,
(5 -> 9 -> 8 -> 11)   , (5 -> 1 -> 11 -> 11)  , (5 -> 4 -> 11 -> 11)  , (6 -> 5 -> 3 -> 1)    , (6 -> 2 -> 5 -> 1)    , (6 -> 5 -> 1 -> 2)    ,
(6 -> 5 -> 1 -> 3)    , (6 -> 2 -> 5 -> 3)    , (6 -> 5 -> 4 -> 4)    , (6 -> 2 -> 5 -> 4)    , (6 -> 5 -> 9 -> 5)    , (6 -> 5 -> 9 -> 7)    ,
(6 -> 5 -> 3 -> 8)    , (6 -> 5 -> 4 -> 8)    , (6 -> 5 -> 9 -> 8)    , (6 -> 5 -> 4 -> 9)    , (6 -> 2 -> 5 -> 9)    , (6 -> 2 -> 11 -> 9)   ,
(6 -> 5 -> 1 -> 11)   , (6 -> 5 -> 4 -> 11)   , (6 -> 2 -> 11 -> 11)  , (7 -> 5 -> 3 -> 1)    , (7 -> 7 -> 3 -> 1)    , (7 -> 10 -> 3 -> 1)   ,
(7 -> 7 -> 5 -> 1)    , (7 -> 3 -> 8 -> 1)    , (7 -> 7 -> 10 -> 1)   , (7 -> 3 -> 1 -> 2)    , (7 -> 5 -> 1 -> 2)    , (7 -> 10 -> 1 -> 2)   ,
(7 -> 3 -> 8 -> 2)    , (7 -> 3 -> 1 -> 3)    , (7 -> 5 -> 1 -> 3)    , (7 -> 10 -> 1 -> 3)   , (7 -> 7 -> 5 -> 3)    , (7 -> 7 -> 7 -> 3)    ,
(7 -> 10 -> 7 -> 3)   , (7 -> 7 -> 10 -> 3)   , (7 -> 5 -> 4 -> 4)    , (7 -> 10 -> 4 -> 4)   , (7 -> 7 -> 5 -> 4)    , (7 -> 7 -> 10 -> 4)   ,
(7 -> 7 -> 7 -> 5)    , (7 -> 10 -> 7 -> 5)   , (7 -> 3 -> 8 -> 5)    , (7 -> 5 -> 9 -> 5)    , (7 -> 10 -> 9 -> 5)   , (7 -> 7 -> 7 -> 7)    ,
(7 -> 10 -> 7 -> 7)   , (7 -> 5 -> 9 -> 7)    , (7 -> 10 -> 9 -> 7)   , (7 -> 7 -> 10 -> 7)   , (7 -> 5 -> 3 -> 8)    , (7 -> 7 -> 3 -> 8)    ,
(7 -> 10 -> 3 -> 8)   , (7 -> 5 -> 4 -> 8)    , (7 -> 10 -> 4 -> 8)   , (7 -> 3 -> 8 -> 8)    , (7 -> 5 -> 9 -> 8)    , (7 -> 10 -> 9 -> 8)   ,
(7 -> 5 -> 4 -> 9)    , (7 -> 10 -> 4 -> 9)   , (7 -> 7 -> 5 -> 9)    , (7 -> 7 -> 10 -> 9)   , (7 -> 7 -> 7 -> 10)   , (7 -> 10 -> 7 -> 10)  ,
(7 -> 3 -> 8 -> 10)   , (7 -> 3 -> 1 -> 11)   , (7 -> 5 -> 1 -> 11)   , (7 -> 10 -> 1 -> 11)  , (7 -> 5 -> 4 -> 11)   , (7 -> 10 -> 4 -> 11)  ,
```

```
(7 -> 3 -> 8 -> 11)   , (8 -> 1 -> 3 -> 1)    , (8 -> 5 -> 3 -> 1)    , (8 -> 10 -> 3 -> 1)   , (8 -> 2 -> 5 -> 1)    , (8 -> 8 -> 5 -> 1)    ,
(8 -> 8 -> 8 -> 1)    , (8 -> 8 -> 10 -> 1)  , (8 -> 5 -> 1 -> 2)    , (8 -> 8 -> 1 -> 2)    , (8 -> 10 -> 1 -> 2)   , (8 -> 8 -> 8 -> 2)    ,
(8 -> 5 -> 1 -> 3)    , (8 -> 8 -> 1 -> 3)   , (8 -> 10 -> 1 -> 3)   , (8 -> 2 -> 5 -> 3)   , (8 -> 8 -> 5 -> 3)    , (8 -> 10 -> 7 -> 3)   ,
(8 -> 8 -> 10 -> 3)   , (8 -> 5 -> 4 -> 4)   , (8 -> 10 -> 4 -> 4)   , (8 -> 2 -> 5 -> 4)   , (8 -> 8 -> 5 -> 4)    , (8 -> 8 -> 10 -> 4)   ,
(8 -> 1 -> 2 -> 5)    , (8 -> 8 -> 2 -> 5)   , (8 -> 10 -> 7 -> 5)   , (8 -> 8 -> 8 -> 5)   , (8 -> 5 -> 9 -> 5)    , (8 -> 10 -> 9 -> 5)   ,
(8 -> 11 -> 9 -> 5)   , (8 -> 10 -> 7 -> 7)  , (8 -> 5 -> 9 -> 7)    , (8 -> 10 -> 9 -> 7)   , (8 -> 11 -> 9 -> 7)   , (8 -> 8 -> 10 -> 7)   ,
(8 -> 1 -> 3 -> 8)    , (8 -> 5 -> 3 -> 8)   , (8 -> 10 -> 3 -> 8)   , (8 -> 5 -> 4 -> 8)   , (8 -> 10 -> 4 -> 8)   , (8 -> 8 -> 8 -> 8)    ,
(8 -> 5 -> 9 -> 8)    , (8 -> 10 -> 9 -> 8)  , (8 -> 11 -> 9 -> 8)   , (8 -> 5 -> 4 -> 9)   , (8 -> 10 -> 4 -> 9)   , (8 -> 2 -> 5 -> 9)    ,
(8 -> 8 -> 5 -> 9)    , (8 -> 8 -> 10 -> 9)  , (8 -> 1 -> 11 -> 9)   , (8 -> 2 -> 11 -> 9)  , (8 -> 8 -> 11 -> 9)   , (8 -> 11 -> 11 -> 9)  ,
(8 -> 10 -> 7 -> 10)  , (8 -> 8 -> 8 -> 10)  , (8 -> 5 -> 1 -> 11)   , (8 -> 8 -> 1 -> 11)   , (8 -> 10 -> 1 -> 11)  , (8 -> 1 -> 2 -> 11)   ,
(8 -> 8 -> 2 -> 11)   , (8 -> 5 -> 4 -> 11)  , (8 -> 10 -> 4 -> 11)  , (8 -> 8 -> 8 -> 11)   , (8 -> 1 -> 11 -> 11)  , (8 -> 2 -> 11 -> 11)  ,
(8 -> 8 -> 11 -> 11)  , (8 -> 11 -> 11 -> 11), (9 -> 5 -> 3 -> 1)    , (9 -> 7 -> 3 -> 1)    , (9 -> 7 -> 5 -> 1)    , (9 -> 8 -> 5 -> 1)    ,
(9 -> 8 -> 8 -> 1)    , (9 -> 7 -> 10 -> 1)  , (9 -> 8 -> 10 -> 1)  , (9 -> 5 -> 1 -> 2)    , (9 -> 8 -> 1 -> 2)    , (9 -> 8 -> 8 -> 2)    ,
(9 -> 5 -> 1 -> 3)    , (9 -> 8 -> 1 -> 3)   , (9 -> 7 -> 5 -> 3)    , (9 -> 8 -> 5 -> 3)    , (9 -> 7 -> 7 -> 3)    , (9 -> 7 -> 10 -> 3)   ,
(9 -> 8 -> 10 -> 3)   , (9 -> 5 -> 4 -> 4)   , (9 -> 7 -> 5 -> 4)    , (9 -> 8 -> 5 -> 4)    , (9 -> 7 -> 10 -> 4)   , (9 -> 8 -> 10 -> 4)   ,
(9 -> 8 -> 2 -> 5)    , (9 -> 7 -> 7 -> 5)   , (9 -> 8 -> 8 -> 5)    , (9 -> 5 -> 9 -> 5)    , (9 -> 7 -> 7 -> 7)    , (9 -> 5 -> 9 -> 7)    ,
(9 -> 7 -> 10 -> 7)   , (9 -> 8 -> 10 -> 7)  , (9 -> 5 -> 3 -> 8)    , (9 -> 7 -> 3 -> 8)    , (9 -> 5 -> 4 -> 8)    , (9 -> 8 -> 8 -> 8)    ,
(9 -> 5 -> 9 -> 8)    , (9 -> 5 -> 4 -> 9)   , (9 -> 7 -> 5 -> 9)    , (9 -> 8 -> 5 -> 9)    , (9 -> 7 -> 10 -> 9)   , (9 -> 8 -> 10 -> 9)   ,
(9 -> 8 -> 11 -> 9)   , (9 -> 7 -> 7 -> 10)  , (9 -> 8 -> 8 -> 10)   , (9 -> 5 -> 1 -> 11)   , (9 -> 8 -> 1 -> 11)   , (9 -> 8 -> 2 -> 11)   ,
(9 -> 5 -> 4 -> 11)   , (9 -> 8 -> 8 -> 11)  , (9 -> 8 -> 11 -> 11) , (10 -> 1 -> 3 -> 1)   , (10 -> 7 -> 3 -> 1)   , (10 -> 7 -> 5 -> 1)   ,
(10 -> 9 -> 5 -> 1)   , (10 -> 3 -> 8 -> 1)  , (10 -> 4 -> 8 -> 1)   , (10 -> 9 -> 8 -> 1)   , (10 -> 7 -> 10 -> 1)  , (10 -> 3 -> 1 -> 2)   ,
(10 -> 3 -> 8 -> 2)   , (10 -> 4 -> 8 -> 2)  , (10 -> 9 -> 8 -> 2)   , (10 -> 3 -> 1 -> 3)   , (10 -> 7 -> 5 -> 3)   , (10 -> 9 -> 5 -> 3)   ,
(10 -> 7 -> 7 -> 3)   , (10 -> 9 -> 7 -> 3)  , (10 -> 7 -> 10 -> 3)  , (10 -> 4 -> 4 -> 4)   , (10 -> 7 -> 5 -> 4)   , (10 -> 9 -> 5 -> 4)   ,
(10 -> 7 -> 10 -> 4)  , (10 -> 1 -> 2 -> 5)  , (10 -> 7 -> 7 -> 5)   , (10 -> 9 -> 7 -> 5)   , (10 -> 3 -> 8 -> 5)   , (10 -> 4 -> 8 -> 5)   ,
(10 -> 9 -> 8 -> 5)   , (10 -> 4 -> 9 -> 5)  , (10 -> 7 -> 7 -> 7)   , (10 -> 9 -> 7 -> 7)   , (10 -> 4 -> 9 -> 7)   , (10 -> 7 -> 10 -> 7)  ,
(10 -> 1 -> 3 -> 8)   , (10 -> 7 -> 3 -> 8)  , (10 -> 4 -> 4 -> 8)   , (10 -> 3 -> 8 -> 8)   , (10 -> 4 -> 8 -> 8)   , (10 -> 9 -> 8 -> 8)   ,
(10 -> 4 -> 9 -> 8)   , (10 -> 4 -> 4 -> 9)  , (10 -> 7 -> 5 -> 9)   , (10 -> 9 -> 5 -> 9)   , (10 -> 7 -> 10 -> 9)  , (10 -> 1 -> 11 -> 9)  ,
(10 -> 4 -> 11 -> 9)  , (10 -> 7 -> 7 -> 10) , (10 -> 9 -> 7 -> 10)  , (10 -> 3 -> 8 -> 10)  , (10 -> 4 -> 8 -> 10)  , (10 -> 9 -> 8 -> 10)  ,
(10 -> 3 -> 1 -> 11)  , (10 -> 1 -> 2 -> 11) , (10 -> 4 -> 4 -> 11)  , (10 -> 3 -> 8 -> 11)  , (10 -> 4 -> 8 -> 11)  , (10 -> 9 -> 8 -> 11)  ,
(10 -> 1 -> 11 -> 11) , (10 -> 4 -> 11 -> 11), (11 -> 9 -> 5 -> 1)   , (11 -> 9 -> 8 -> 1)   , (11 -> 9 -> 8 -> 2)   , (11 -> 9 -> 5 -> 3)   ,
(11 -> 9 -> 7 -> 3)   , (11 -> 9 -> 5 -> 4)  , (11 -> 9 -> 7 -> 5)   , (11 -> 9 -> 8 -> 5)   , (11 -> 11 -> 9 -> 5)  , (11 -> 9 -> 7 -> 7)   ,
(11 -> 11 -> 9 -> 7)  , (11 -> 9 -> 8 -> 8)  , (11 -> 11 -> 9 -> 8)  , (11 -> 9 -> 5 -> 9)   , (11 -> 11 -> 11 -> 9) , (11 -> 9 -> 7 -> 10)  ,
(11 -> 9 -> 8 -> 10)  , (11 -> 9 -> 8 -> 11) , (11 -> 11 -> 11 -> 11),
```

Матриці досяжності та сильної зв’язності:

```
=== Reachability matrix ===
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
```

```
=== Strong connectivity ===
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1
```

Компоненти сильної зв’язності:

```
Strong connectivity components:
Component 1: nodes [1, 2, 3, 4, 5, 7, 8, 9, 10, 11]

=== Condensation graph ===
0
```

Граф конденсації:

Висновок:
Засвоїв теоретичний матеріал лекцій збільшив практичний досвід у створенні, відображені напрямлених і ненапрямених графів, та набув навичок у їх аналізуванні. Покращив навички роботи з графічними вікнами та функціями для них. Збільшив досвід роботи з матрицями та створення функцій для їх аналізу.